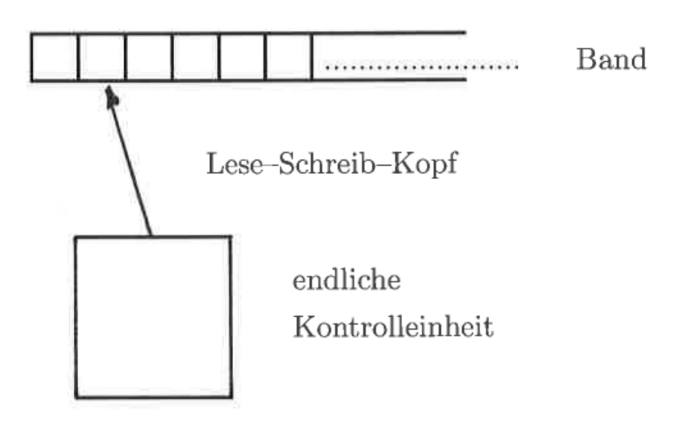
Turing–Maschinen und Unentscheidbarkeit

1.4. Turing–Maschinen und rekursiv aufzählbare Sprachen

Um zu beweisen, daß eine bestimmte Funktion "berechenbar" ist, oder ein gewisses Minimum an Zeit zur Berechnung benötigt (untere Komplexitätsschranke), brauchen wir ein Maschinenmodell, das zwar so allgemein aber einfacher als eine RAM oder RASP ist. Dieses Modell ist die Turing–Maschine [1)].

Definition 1.4.1: Eine *Turing–Maschine M* können wir uns folgendermaßen vorstellen:



Sie besteht aus einer *endlichen Kontrolleinheit*, welche im Inneren eine endliche Anzahl von Zuständen aus einer *Zustandsmenge* $Q = \{q_0, \ldots, q_n\}$ annehmen kann, einem in eine Richtung unendlichen *Band*, welches in Zellen eingeteilt ist, deren jede ein Symbol aus einem endlichen *Alphabet von Bandsymbolen* $\Gamma = \{\gamma_0, \ldots, \gamma_m\}$ speichern kann — γ_0 wird dabei als das Blank Symbol $\sqcup$ aufgefaßt und eine Untermenge Σ von $\Gamma \setminus \{\sqcup\}$ ist das *Eingabealphabet* — , sowie aus einem *Lese–Schreib–Kopf*, welcher immer auf eine Bandzelle zeigt.

Wie funktioniert nun die Turing–Maschine M? Wenn die Maschine zu arbeiten beginnt, ist die endliche Kontrolleinheit im Anfangszustand q_0, und bis auf endlich viele Zellen am linken Ende des Bandes sind alle Bandzellen mit $\sqcup$'s belegt. In diskreten Zeitschritten, abhängig vom Zustand und vom Bandsymbol, auf welches der LS–Kopf gerade zeigt (Symbol in der *Arbeitszelle*), geschieht folgendes:

[1)] A.M. Turing, "On computable numbers, with an application to the Entscheidungsproblem", *Proc. London Math. Soc.*, series 2, 42:230–265 (1936/37). Corrections in *Proc. London Math. Soc.*, 43:544–546 (1937)

- M nimmt einen neuen Zustand an,
- schreibt ein Bandsymbol in die Arbeitszelle, und
- bewegt den LS–Kopf um eine Position nach rechts (R) oder nach links (L) oder läßt ihn stationär (S).

Erreicht dabei M einen Zustand aus einer Menge von *Endzuständen* F (Teilmenge der Zustandsmenge), so terminiert die Berechnung erfolgreich, M *akzeptiert* die Eingabe. Ansonsten fährt M in der Berechnung so lange fort, als zum jeweiligen Zustand und Symbol in der Arbeitszelle ein Nachfolgerzustand erklärt ist. Eine Berechnung, welche nicht erfolgreich terminiert, kann also auch unendliche Länge haben. ■

Beispiel 1.4.1: Als Beispiel betrachten wir die Turing–Maschine M_1, welche erkennt, ob eine endliche Folge x über $\{0,1\}$ die Form $0^n 1^n$ hat, für eine natürliche Zahl n. Bei Anfang der Berechnung steht die Folge x in den ersten Zellen des Bandes von M_1, gefolgt von einer unendlichen Folge von $\sqcup$'s. M_1 führt folgende Berechnungsschleife aus:

> M_1 ersetzt die am weitesten links stehende 0 durch X, bewegt den LS–Kopf nach rechts bis zur am weitesten links stehenden 1, ersetzt diese durch Y, bewegt den LS–Kopf nach links bis zum ersten X und anschließend um eine Position nach rechts zur am weitesten links stehenden 0 und wiederholt die Schleife.

Findet nun M_1 bei der Suche nach einer 1 anstatt dessen ein $\sqcup$, so stoppt M_1 die Berechnung und gibt "nein" als Antwort. Falls M_1, nachdem es eine 1 durch Y ersetzt hat, keine weitere 0 mehr findet, so prüft es, ob noch 1en vorhanden sind, und gibt in diesem Fall die Antwort "nein", andernfalls die Antwort "ja".

Etwas formaler könnten wir M_1 wie folgt angeben:

Zustandsmenge .... $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
Eingabealphabet ... $\Sigma = \{0, 1\}$,
Bandalphabet ..... $\Gamma = \{\sqcup, 0, 1, X, Y\}$,
Endzustände ...... $F = \{q_4\}$,

Überführungsfunktion δ:

Zustand \ Symbol	⊔	0	1	X	Y
q_0	—	(q_1, X, R)	—	—	(q_3, Y, R)
q_1	—	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)
q_2	—	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)
q_3	$(q_4, \sqcup, R)$	—	—	—	(q_3, Y, R)
q_4	—	—	—	—	—

Jeden Zustand könnte man sich vorstellen als eine Anweisung oder eine Gruppe von Anweisungen in einem Programm.

- Der Zustand q_0 wird am Start angenommen und auch jeweils unmittelbar vor der Ersetzung der am weitesten links stehenden 0 durch X. Findet M_1 eine 0 vor, so ersetzt es diese durch X, geht über in q_1, und bewegt den LS–Kopf eine Position nach rechts. Findet q_0 ein Y vor, so nimmt M_1 den Zustand q_3 an.

- q_1 wird dazu benutzt, um von links nach rechts nach der ersten 1 zu suchen. Findet M_1 eine 1, so wird sie durch Y ersetzt, und M_1 nimmt den neuen Zustand q_2 an. Wird vor einer 1 ein $\sqcup$ oder ein X gefunden, so bricht die Berechnung ab, und die Eingabe wird zurückgewiesen.

- q_2 wird benutzt um von rechts nach links nach dem ersten X zu suchen. Wird ein X gefunden, so nimmt M_1 den Zustand q_0 an und bewegt den LS–Kopf eine Position nach rechts.

- Im Zustand q_3 wird der LS–Kopf über Y's hinweg nach rechts bewegt um zu prüfen, ob noch 1en auf dem Band vorkommen. Ist das erste Symbol nach den Y's ein $\sqcup$, so nimmt M_1 den Zustand q_4 an und die Eingabefolge wird damit akzeptiert. Andernfalls wird die Eingabefolge zurückgewiesen. ∎

Definition 1.4.1: (Fortsetzung) Etwas formaler wird eine *Turing–Maschine* (kurz *TM*) so angegeben wie im obigen Beispiel, nämlich als ein 6–tupel $(Q, \Sigma, \Gamma, q_0, F, \delta)$, wobei Q die Zustandsmenge ist, Σ das Eingabealphabet, Γ das Bandalphabet, q_0 der Startzustand, F die Menge der Endzustände und δ die Überführungsfunktion. δ ist eine partielle Funktion von $Q \times \Gamma$ in $Q \times \Gamma \times \{L, R, S\}$. ∎

Definition 1.4.2: Sei M ein TM. Eine *Konfiguration* von M legt den Zustand von M zu einem bestimmten Zeitpunkt fest, also eine Folge von Bandsymbolen $\alpha_1 \dots \alpha_k$, welche links vom LS–Kopf auf dem Band stehen, eine Folge von Bandsymbolen $\alpha_{k+1} \dots \alpha_l$, welche rechts vom LS–Kopf auf dem Band stehen einschließlich dem Symbol in der Arbeitszelle (die unendlich vielen $\sqcup$'s am rechten Bandende werden dabei vernachlässigt), sowie den Zustand q der endlichen Kontrolleinheit. Alle anderen Bandsymbole rechts von der Position l sind $\sqcup$. Wir schreiben dafür

$$\alpha_1 \dots \alpha_k q \alpha_{k+1} \dots \alpha_l.$$

Also $\alpha_1 \dots \alpha_k q \alpha_{k+1} \dots \alpha_l$ und $\alpha_1 \dots \alpha_k q \alpha_{k+1} \dots \alpha_l \sqcup$ sind nur zwei verschiedene Notationen für ein und dieselbe Konfiguration.

Eine Konfiguration $\beta_1 \dots \beta_k q \beta_{k+1} \dots \beta_m$ geht aus $\alpha_1 \dots \alpha_l p \alpha_{l+1} \dots \alpha_m$ hervor,

$$\alpha_1 \dots \alpha_l p \alpha_{l+1} \dots \alpha_m \quad \vdash \quad \beta_1 \dots \beta_k q \beta_{k+1} \dots \beta_m,$$

wenn $\alpha_i = \beta_i$ für $i \in \{1, \dots, l, l+2, \dots, m\}$ und

$$\delta(p, \alpha_{l+1}) = (q, \beta_{k+1}, S) \quad \text{und} \quad k = l,$$

oder

$$\delta(p, \alpha_{l+1}) = (q, \beta_k, R) \quad \text{und} \quad k = l+1,$$

oder

$$\delta(p, \alpha_{l+1}) = (q, \beta_{k+2}, L) \quad \text{und} \quad k = l-1.$$

Eine *Berechnung* der Turing–Maschine M ist eine Folge von Konfigurationen, wobei jeweils die $(i+1)$–te Konfiguration aus der i–ten Konfiguration hervorgeht.

M akzeptiert das Eingabewort $x \in \Sigma^*$, wenn die Berechnung ausgehend von der Konfiguration $q_0 x$ nach endlich vielen Schritten zu einer Konfiguration führt, in welcher der Zustand q ein Element von F ist. ■

Beispiel 1.4.2: Auf der Eingabe 0011 etwa führt die Turing–Maschine M_1 folgende Berechnung aus:

$$\begin{aligned}
&q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash \\
&q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash \\
&X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash \\
&X X Y Y q_3 \vdash X X Y Y \sqcup q_4.
\end{aligned}$$

Die Eingabe 0011 wird also von M_1 akzeptiert. ■

Def. 1.4.3: Sei M eine Turing–Maschine . $L(M)$, die *von M akzeptierte Sprache*, ist die Menge aller Eingabewörter x (Folgen von Symbolen aus dem Eingabealphabet Σ), welche von M akzeptiert werden. ■

Also $L(M_1) = \{ \ 0^n 1^n \ \mid \ n \geq 1 \ \}$.

Def. 1.4.4: Eine Sprache L (Menge von Wörtern über einem Alphabet Σ) heißt *rekursiv aufzählbar (r.a.)*, wenn $L = L(M)$ für eine Turing–Maschine M. Eine r.a. Sprache L, welche von einer Turing–Maschine M akzeptiert wird, die auf jedem Eingabewort stoppt, heißt eine *rekursive* Sprache. In diesem Fall sagt man, M *erkennt* L. ■

4.1. Akzeptierungsproblem und Halteproblem für Turing–Maschinen

Unter einem *(Entscheidungs–) Problem* verstehen wir eine Frage der Art *"Ist die natürliche Zahl n ein Quadrat?"* Für eine Festlegung des Parameters n ist die Antwort entweder *"ja"* oder *"nein"*. Eine *Instanz* eines Problems ist eine Liste von Argumenten, ein Argument für jeden Parameter des Problems. So ist etwa (15) eine Instanz des obigen Problems.

Einem Problem P mit Parametern $p_1, \ldots, p_k$ kann auf offensichtliche Weise eine Sprache L_P wie folgt zugeordnet werden:

$$L_P = \{(n_1, \ldots, n_k) \mid \text{die Antwort auf die Instanz } (n_1, \ldots, n_k) \text{ des Problems } P \text{ ist "ja"}\}.$$

Definition 4.1.1: Ein Problem P, dessen Sprache L_P rekursiv ist, heißt *entscheidbar*. Andernfalls heißt P *unentscheidbar*. Ist L_P r.a., so sagt man auch P wäre *semi–entscheidbar*. ■

Definition 4.1.2: Zunächst bringen wir jede Turing–Maschine in eine gewisse kanonische Form, man spricht dabei von der *Kodierung* einer Turing–Maschine. Sei M eine Turing–Maschine mit Zustandsmenge $\{q_1, \ldots, q_n\}$, Eingabealphabet $\{0, 1\}$, Bandalphabet $\{0, 1, \sqcup\}$, Anfangszustand q_1, Endzustandsmenge $F = \{q_2\}$ und Überführungsfunktion δ. Das stellt keine Einschränkung dar, da jedes Alphabet Σ in $\{0, 1\}$ kodiert werden kann. Wir bezeichnen die Bewegungsrichtungen "links", "rechts" mit D_1, D_2 (der Einfachheit halber lassen wir nicht zu, daß der LS–Kopf stationär bleibt; diese Situation kann aber immer durch eine Folge von rechts–links Bewegungen simuliert werden), und die Symbole 0,1,$\sqcup$ mit X_1, X_2, X_3. Eine allgemeine Operation von M hat also die Gestalt

$$\delta(q_i, X_j) = (q_k, X_l, D_m).$$

Diese Operation wird kodiert als

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m. \qquad (4.1.1)$$

Die Turing–Maschine M wird kodiert als

$$111 \; code_1 \; 11 \; code_2 \; 11 \; \ldots \; 11 \; code_r \; 111, \qquad (4.1.2)$$

wobei jeder $code_i$ die Form (4.1.1) hat und jede Operation von M von einem $code_i$ kodiert ist. Die (bis auf Umordnung der $code_i$'s eindeutig bestimmte) Kodierung von M wird mit $\langle M \rangle$ bezeichnet. ■

Beispiel 4.1.1: Als Beispiel einer solchen Kodierung betrachten wir die Turing–Maschine M mit $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$, $F = \{q_2\}$, und $\delta(q_1, 1) = (q_3, 0, R)$, $\delta(q_3, 0) = (q_1, 1, R)$, $\delta(q_3, 1) = (q_2, 0, R)$, $\delta(q_3, \sqcup) = (q_3, 1, L)$. Eine mögliche Kodierung von M (Version von $\langle M \rangle$) ist

$$11101001000101001100010101001001100010010010100110001000100010010111. \quad ■$$

Es ist also klar, daß die (Kodes von) Turing–Maschinen in einer Reihe $M_1, M_2, M_3, \ldots$ angeordnet werden können (etwa der Länge nach und alphabetisch bei gleicher Länge). Ebenso können die Wörter in $\{0, 1\}^*$ in einer Reihe $w_1, w_2, w_3, \ldots$ angeordnet werden. Für $i, j \in \mathbf{N}$ definieren wir nun

$$a_{ij} = \begin{cases} 0, & \text{falls } w_i \in L(M_j), \\ 1, & \text{sonst.} \end{cases}$$

Auf diese Weise erhalten wir eine nach unten und rechts unendliche Matrix

Wörter \ TM	M_1	M_2	M_3	$\ldots$
w_1	a_{11}	a_{12}	a_{13}	$\ldots$
w_2	a_{21}	a_{22}	a_{23}	$\ldots$
w_3	a_{31}	a_{32}	a_{33}	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Definition 4.1.3: Wir benutzen die Diagonalelemente in dieser Matrix zur Definition der *Diagonalsprache*

$$L_d = \{ \, w_j \mid a_{jj} = 1 \, \}.$$

Also $w_j \in L_d \iff w_j \notin L(M_j)$. ■

Satz 4.1.1: *Die Diagonalsprache L_d ist nicht r.a.*

Beweis: Angenommen es gäbe eine Turing–Maschine M_j, welche L_d akzeptiert. Daraus würde sich folgende Kontradiktion ergeben: falls $w_j \in L_d$, dann $a_{jj} = 1$, also $w_j \notin L(M_j)$. Andererseits, falls $w_j \notin L_d$, dann $a_{jj} = 0$, also $w_j \in L(M_j)$. $L(M_j)$ wäre also verschieden von L_d. ∎

Definition 4.1.4: Das *Akzeptierungsproblem für Turing–Maschinen* ist das Problem, für eine gegebene Turing–Maschine M (mit Bandalphabet $\{0, 1, \sqcup\}$) und ein gegebenes Wort $w \in \{0, 1\}^*$ zu entscheiden, ob M das Wort w akzeptiert. Die zum Akzeptierungsproblem gehörige Sprache heißt die *universelle Sprache.* Sie ist definiert als

$$L_u = \{ \ \langle M \rangle w \ \mid \ M \text{ akzeptiert } w \ \}.$$ ∎

L_u ist universell in dem Sinn, daß die Frage, ob eine Folge w in $\{0, 1\}^*$ von einer Turing–Maschine M (o.B.d.A. mit Bandalphabet $\{0, 1, \sqcup\}$) akzeptiert wird, äquivalent ist zur Frage, ob $\langle M \rangle w \in L_u$.

Satz 4.1.2: *L_u ist r.a.*

Beweis: Wir konstruieren eine Turing–Maschine mit 3 Bändern M_u^3, welche L_u akzeptiert. Das erste Band von M_u^3 ist das Eingabeband und enthält also eine Instanz $\langle M \rangle w$ des Akzeptierungsproblems. Die Operationen der Turing–Maschine M sind zwischen den ersten zwei Blöcken von drei 1en auf dem ersten Band kodiert. Das zweite Band von M_u^3 simuliert das Band von M. Auf dem dritten Band von M_u^3 wird der Zustand q_i von M als 0^i abgespeichert. Die Turing–Maschine M_u^3 funktioniert folgendermaßen:

(1) Überprüfe den Inhalt von Band 1 darauf hin, ob er mit einem Wort der Form (4.1.2) beginnt und es keine zwei Operationskodes gibt, welche mit 0^i10^j1 für dasselbe i und j beginnen. Weiters prüfe für jeden Operationskode $0^i10^j10^k10^l10^m$ ob $1 \leq j \leq 3$, $1 \leq l \leq 3$ und $1 \leq m \leq 2$. Bei diesen Berechnungen kann Band 3 als Arbeitsband benutzt werden.

(2) Schreibe w, den Teil der Eingabe nach dem zweiten Block von drei 1en, auf Band 2. Schreibe 0 (die Kodierung für den Zustand q_1) auf Band 3. Alle LS–Köpfe werden am linken Bandende positioniert. Diese Symbole können markiert werden, sodaß sie leicht wieder aufgefunden werden können.

(3) Enthält Band 3 das Wort 00, den Kode für den Endzustand q_2, so halte und akzeptiere.

(4) Sei X_j das aktuelle Symbol (auf welches der LS–Kopf zeigt) von Band 2 und sei 0^i der Inhalt von Band 3. Suche auf Band 1 von links bis zum zweiten Block von drei 1en nach einem Unterwort mit dem Präfix 110^i10^j1. Wird ein solches Wort nicht gefunden, so halte und weise die Eingabe zurück; M kann keinen nächsten Schritt ausführen und hat nicht akzeptiert. Wird jedoch ein Kode der Form $0^i10^j10^k10^l10^m$ gefunden, so schreibe 0^k auf Band 3, schreibe X_l auf die aktuelle Bandzelle in Band 2 und bewege den LS–Kopf 2 in die Richtung D_m. Fahre fort mit Schritt (3).

Offensichtlich akzeptiert M_u^3 das Eingabewort $\langle M\rangle w$ genau dann, wenn M das Wort w akzeptiert. Hält M nicht auf w oder hält es ohne zu akzeptieren, so tut M_u^3 dasselbe auf $\langle M\rangle w$. ■

Definition 4.1.5: Da L_u r.a. ist, gibt es eine Turing–Maschine M_u (mit einem Band), welche L_u akzeptiert. Diese (allerdings nicht eindeutig bestimmte) Turing–Maschine M_u heißt die *universelle Turing–Maschine*, da sie die Arbeit jeder Turing–Maschine verrichtet. ■

Die universelle TM L_u ist ein Interpreter für jede TM.

Satz 4.1.3: *L_u ist nicht rekursiv, das Akzeptierungsproblem für Turing–Maschinen ist also unentscheidbar.*

Beweis: Wir nehmen an, es gäbe eine Turing–Maschine M, welche L_u entscheidet (d.h. M stoppt auf jeder Eingabe und akzeptiert L_u). Dann könnten wir L_d wie folgt entscheiden: für eine gegebene Eingabe w bestimmen wir zunächst den Index i, für welchen gilt $w = w_i$. Aus i berechnen wir die Turing–Maschine M_i. Die Folge $\langle M_i\rangle w_i$ geben wir nun als Eingabe an M und akzeptieren w genau dann, wenn M die Folge $\langle M_i\rangle w_i$ nicht akzeptiert, also M_i das Wort w_i nicht akzeptiert. Da wir aber aus (1) wissen, daß L_d nicht rekursiv ist, kann es keinen solchen Entscheidungsalgorithmus geben, also auch keine Turing–Maschine M, die L_u entscheidet. ■

Ein scheinbar einfacher zu behandelndes Problem ist das *Halteproblem "stoppt (hält) die Berechnung der Turing–Maschine M auf dem Eingabewort w?"* Man sieht aber leicht, daß auch das Halteproblem unentscheidbar sein muß, da man andernfalls sofort einen Algorithmus zur Entscheidung des Akzeptierungsproblems konstruieren könnte. Bei genauerer Betrachtung stellt sich sogar heraus, daß es nicht einmal einen Algorithmus gibt, der von einer Turing–Maschine entscheiden könnte, ob sie auf der leeren Eingabe stoppt.

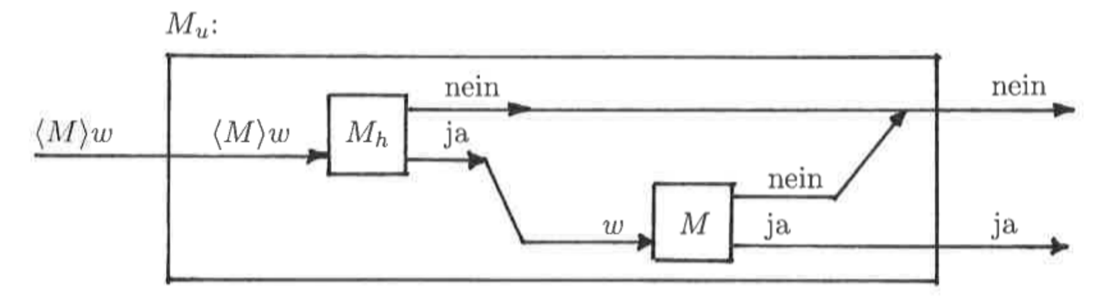
Def. 4.1.6: Mit der selben Bezeichnung wie in Def. 4.1.4 seien

$$L_h = \{ \langle M \rangle w \mid M \text{ stoppt bei Eingabe } w \}, \quad L_{h,\epsilon} = \{ \langle M \rangle \mid M \text{ stoppt bei Eingabe } \epsilon \}.$$

L_h ist die zum *Halteproblem* gehörige Sprache, $L_{h,\epsilon}$ die zum *eingeschränkten Halteproblem* gehörige Sprache. ■

Satz 4.1.4: *L_h ist nicht rekursiv, also das Halteproblem für Turing–Maschinen ist unentscheidbar.*

Beweis: Wäre L_h rekursiv, so wäre auch L_u rekursiv. Wäre nämlich M_h eine immer terminierende Turingmaschine, welche L_h akzeptiert, so wäre M_u eine immer terminierende universelle Turingmaschine:



Dabei genügt es, Eingaben $\langle M \rangle w$ zu betrachten, wobei die TM M genau dann hält, wenn sie die Eingabe akzeptiert. Jede TM kann auf algorithmische Weise in eine solche TM transformiert werden. ■

Auch das eingeschränkte Halteproblem $L_{h,\epsilon}$ ist unentscheidbar, wir können das aber erst später beweisen.

Die üblichen Programmiersprachen sind universell, d.h. sie sind in der Lage, jede Turing–Maschine zu simulieren und umgekehrt.